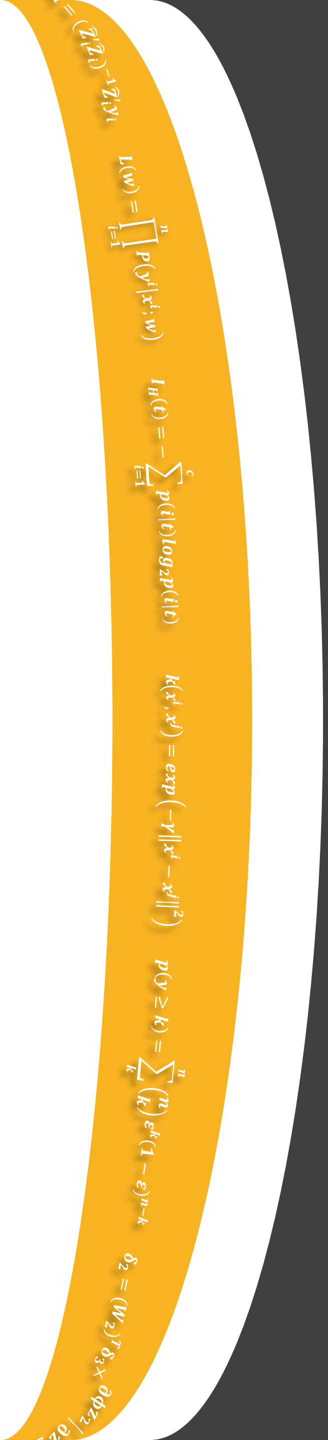
LESSON 2 – INTRODUCTION TO CODING IN PYTHON AND R

Dr. Jack Hong

Adjunct Faculty, Lee Kong Chian School of Business, SMU
Co-founder, Research Room Pte. Ltd.


$= (Z_i'Z_i)^{-1}Z_i'y_i$
$L(w) = \prod_{i=1}^{n} P(y^i|x^i;w)$
$I_H(t) = -\sum_{i=1}^{c} p(i|t)log_2p(i|t)$
$k(x^i,x^j) = exp(-\gamma\|x^i - x^j\|^2)$
$p(y \geq k) = \sum_{k}^{n}\binom{n}{k}\varepsilon^k(1-\varepsilon)^{n-k}$
$\delta_2 = (W_2)^T\delta_3 * \partial\phi z_2 / \partial z_2$

INTRODUCTION TO CODING

WHAT EXACTLY IS CODING?

- Coding is a tradecraft
- Coding is about writing instructions to a computer
  - The computer follows the instructions, takes in data input, and churn out data output
  - A simple mathematical instruction:
    - Instruction: 1 + 1
    - Computer takes in above instruction: X + Y
    - Computer understands that the data input is X = 1, Y = 1
    - Computer returns 2
  - A simple logic instruction:
    - If 3 > 2, then return "Yes, that is correct.", else return "No, that is wrong."
    - Computer takes in the above instruction: if X > Y, then return A, else return B
    - Computer understands that the data input is:
      - X = 3; Y = 2; A = "Yes, that is correct."; B = "No, that is wrong."
    - Computer performs the logic (3 > 2) and determines that it is true
      - It then returns "Yes, that is correct."

CODING INSTRUCTIONS AND LANGUAGE

- A coding language is a piece of software that allows us to write instructions to a computer
- There are many different languages out there, the more popular ones being:
  - C, C++, Java, Python, Ruby
  - All of them are open-source (meaning that the software is free for anyone to use, and the software is maintained by the community)
- Coding principles are the same regardless of the language: master one, master all.
- The main differences between the languages are:
  - Syntax (how instructions should be formatted)
  - Additional functionalities (the respective communities may decide whether a set of functions are integral to the language, and decide to create specialized syntax to invoke them natively)
  - Speed of execution (some languages execute instructions faster than the others)

WHY PYTHON AND R

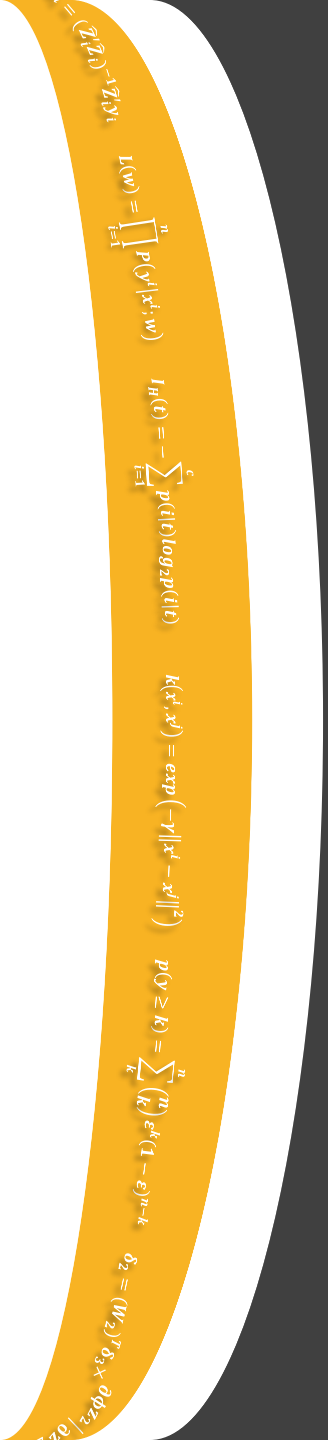
- In this course (and most data science work in general), we choose to learn about Python and R
- The main reasons are:
  - All scientists, social scientists, engineers, and any other scientists who deals with empirical (data) work uses either or both of these languages
    - This implies that both languages are well supported by a large community, larger and more experienced than any software company can recruit on their payroll
    - This also implies that we can find the most updated and comprehensive scientific modules in these languages
    - Easy to find good answers to coding questions (check out stackoverflow)
  - Both languages are easy to learn (no awkward syntax or formatting, sufficient abstraction but not obfuscated)
  - Both languages are free and can be easily installed in most operating systems (Windows, Mac, Linux)
- There are some "purists" out there that ridicule Python and R, and recommends C and Java
  - Most of their arguments are unfounded and misinformed
  - There are no weaknesses that we cannot correct within the Python framework, albeit with a little more effort

WHEN DO WE USE PYTHON OR R?

- Python is powerful enough to do anything we want, in an effective and convenient manner
- However, R is more convenient when it comes to statistical packages for inference
  - E.g. R has modules that output all statistical results useful for inference (i.e. whether X has a significant influence on Y or not)
  - R has a host of specialized statistical treatments that Python does not have modules for (Python's packages are mainly geared towards prediction, not inference)
- The general workflow is thus:
  - Data collection, munging, exploration, and prediction in **Python**
  - Machine learning and Deep learning in **Python**
  - Export the intermediate data from Python to **R** for:
    - Statistical analysis and inference
    - Pretty visualizations

LESSON PATH FOR CODING

- I will orientate you to general coding principles in Python first
- We will start with setting up your workspace
  - Instead of installing Python in your personal computer, we will access it from your browser with
    - Google Colab
    - My JupyterHub server
  - This implies that:
    - We can skip all the installation troubleshooting
    - You can access the programming language as long as you have an internet connection and a web browser
    - You will be using the server's computational power when you execute codes


$= (Z_i'Z_i)^{-1}Z_i'y_i$
$L(w) = \prod_{i=1}^{n} P(y^i|x^i;w)$
$I_H(t) = -\sum_{i=1}^{c} p(i|t)log_2p(i|t)$
$k(x^i,x^j) = exp(-\gamma||x^i - x^j||^2)$
$p(y \geq k) = \sum_{k}^{n} \binom{n}{k} \varepsilon^k(1-\varepsilon)^{n-k}$
$\delta_2 = (W_2)^T\delta_3 * \partial\phi(z_2)/\partial z_2$

INTRODUCTION TO CODING WITH PYTHON

ABOUT PYTHON

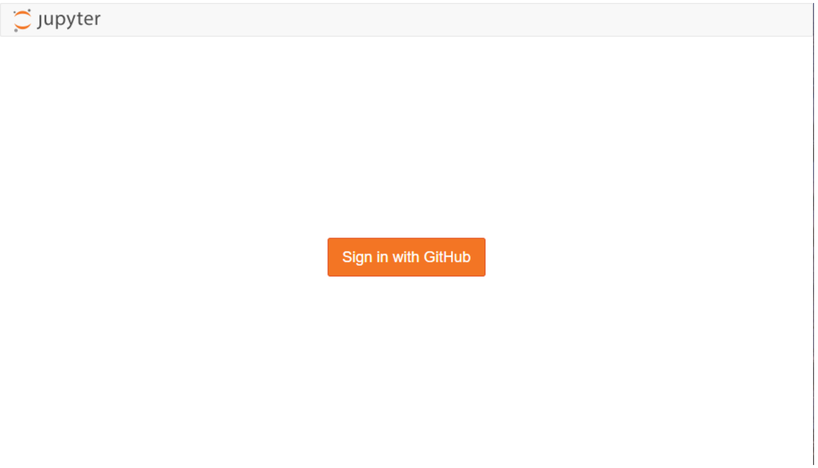
- Python comes with an extremely rich standard library (library is a collection of modules/functions)
  - Made even more comprehensive by a lot of high quality 3rd party packages (a package is another name for module)
- We can create software for any purpose using Python
  - Including writing an Operating System (like windows or Mac) in Python
- Python is a high-level general purpose language designed by Guido van Rossum in the late 1980s
  - Guido is known as the BDFL (Benevolent Dictator for Life) of Python
    - Guido currently spends 50% of his time at Google
  - The name Python was inspired from the comedy Monty Python's Flying Circus
- Python is currently, hands-down, the easiest and most powerful glue language in the coding universe
  - Allows coders to combine different software components together
  - This allows use to replace or enhance parts of Python with more powerful software (e.g. Apache Spark, Apache Kafka, Apache Storm)

SOME POPULAR SCIENTIFIC MODULES IN PYTHON

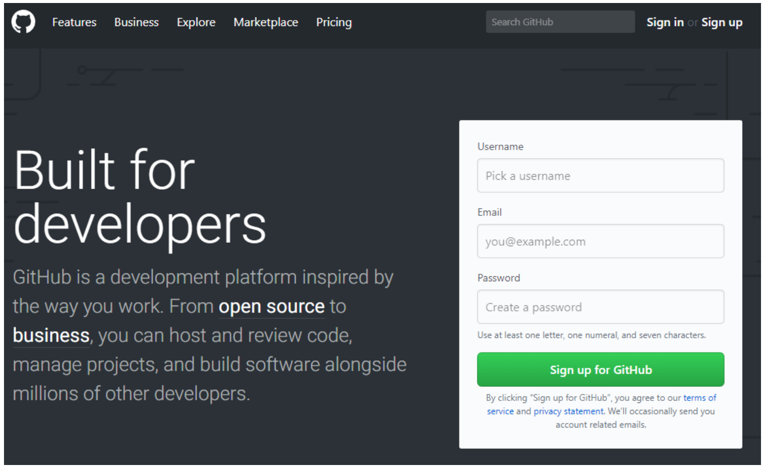
- Numpy
  - Numerical computing
  - The foundation for all other scientific libraries
- Scipy
  - Scientific computing
- Scikit-Learn
  - Statistics and Machine learning
- Pandas
  - Data munging and analysis
- Keras
  - Deep learning

PYTHON HANDS-ON

- We will be using the Jupyter notebook to run both Python and R codes
- Option 1:You can access a Jupyter server through Google Colab
  - https://colab.research.google.com/
- Option 2:You can access my Jupyter server
  - https://sumeru.io/
- For the 2nd option, you will be asked to login via a GitHub account
  - GitHub is a team repository for codes
  - Most, if not all, developers will have a GitHub account
  - You can sign up for a free account at https://github.com



Jupyter Notebook



Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside millions of other developers.

Username

Pick a username

Email

you@example.com

Password

Create a password

Use at least one letter, one numeral, and seven characters.

Sign up for GitHub

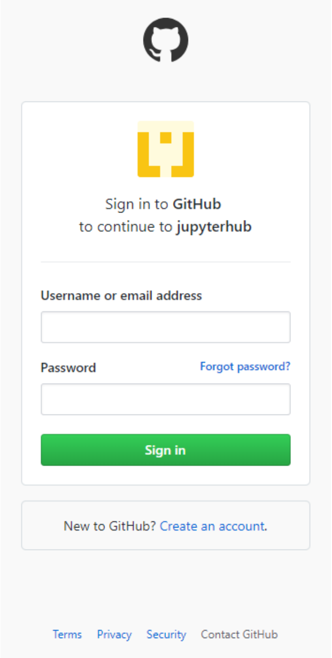
By clicking "Sign up for GitHub", you agree to our terms of service and privacy statement. We'll occasionally send you account related emails.

Github

ACCESSING YOUR NOTEBOOK

- Once you have signed in to https://sumeru.io/ using your GitHub credentials, you will be directed to your private dashboard
- For Google Colab, you will also see your private dashboard with additional tutorials

Sign in to sumeru.io using your GitHub credentials

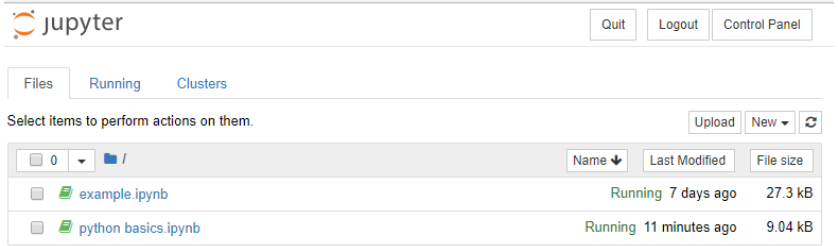


Sign in to GitHub
to continue to jupyterhub

Username or email address

Password

Your private Jupyter Notebook Dashboard

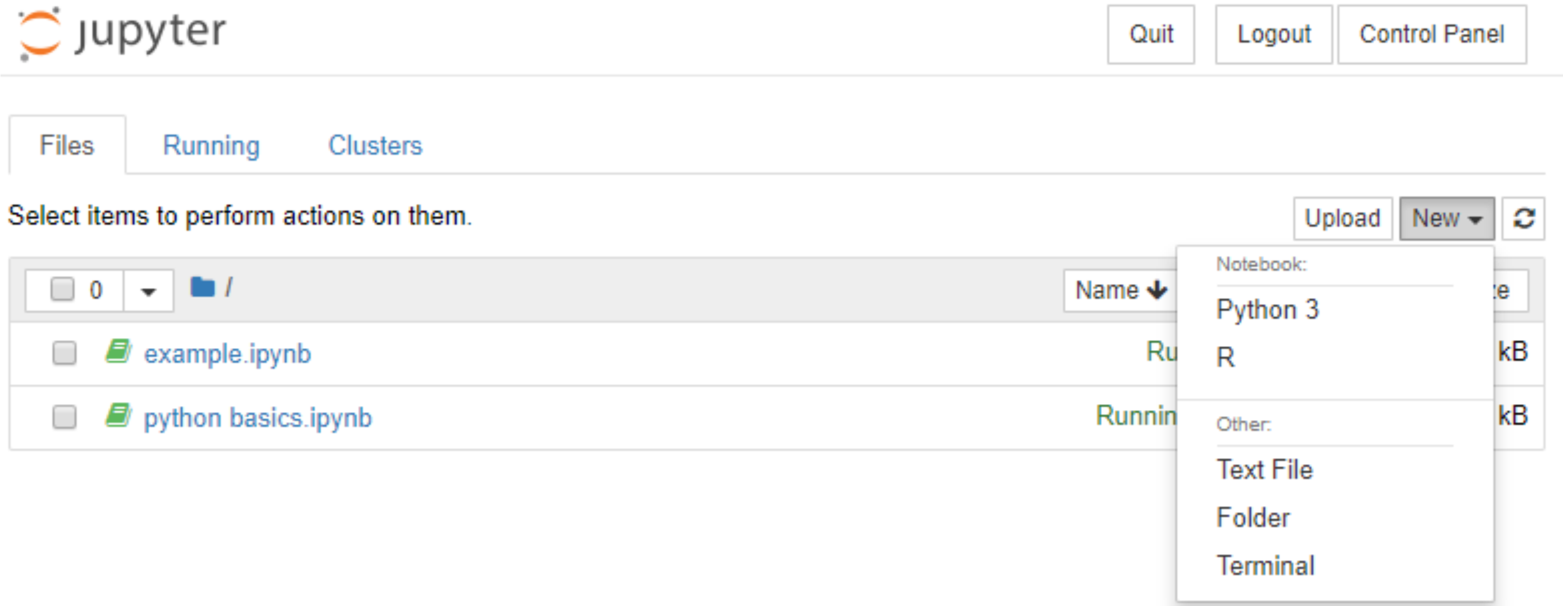


Select items to perform actions on them.

Name		Last Modified	File size
example.ipynb	Running	7 days ago	27.3 kB
python basics.ipynb	Running	11 minutes ago	9.04 kB

USING YOUR DASHBOARD

- You can create new Python or R notebooks by clicking on the "New" button and choosing Python 3 or R
  - On Google Colab, you click on File > New Python 3 Notebook
- All notebooks that you have created will be shown on your dashboard
- All notebooks are automatically saved



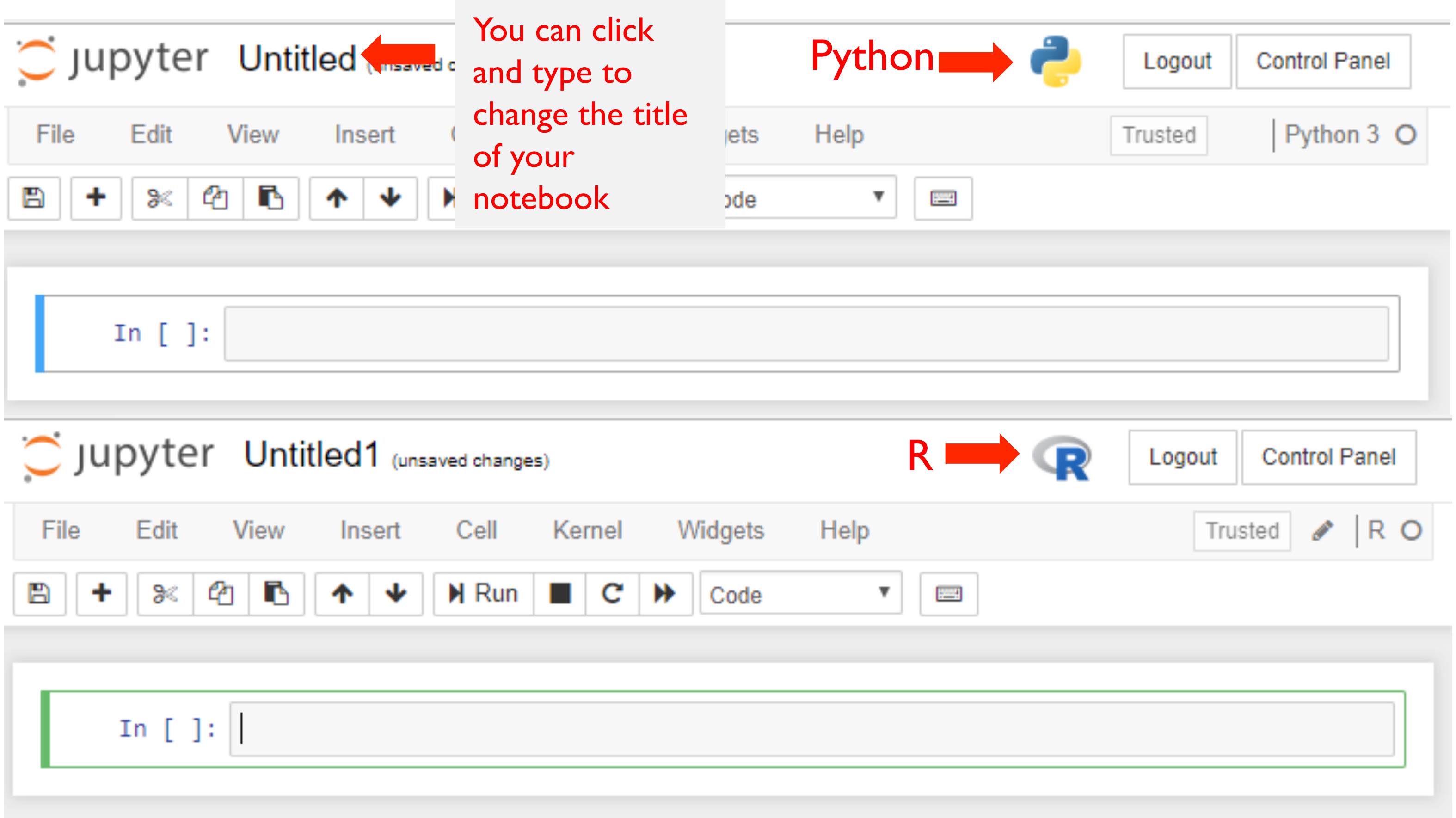
Select items to perform actions on them.

example.ipynb Ru kB

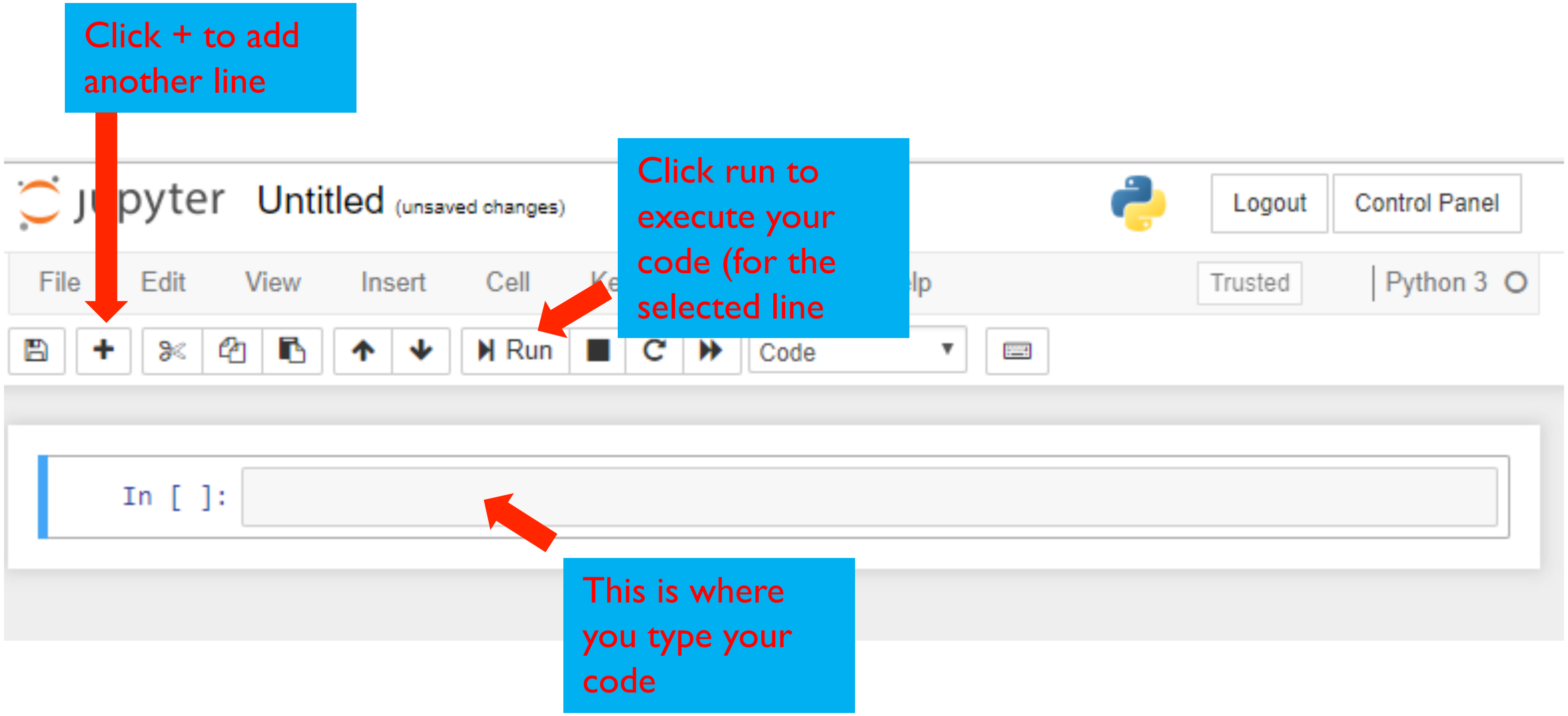
python basics.ipynb Runnin kB

JUPYTER NOTEBOOKS

- Once you create a new notebook, you will be presented with a coding interface for the respective language
- On my JupyterHub, you can check which language the notebook is launched in by looking at the logo on the top right hand side of the browser



USING THE NOTEBOOK



Click + to add another line

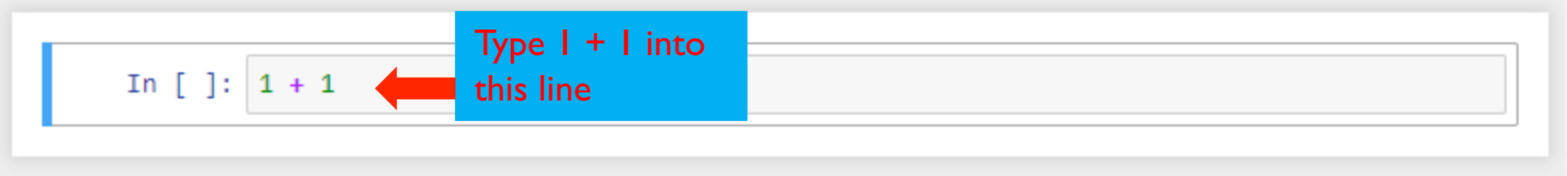
Click run to execute your code (for the selected line

This is where you type your code

- Jupyter notebooks are automatically saved
- Codes in Jupyter notebooks are executed line by line
  - To select a line, simply click on it
  - The shortcut for executing the selected line is ctrl-enter
  - The shortcut for executing the selected line and created a new one underneath it, is alt-enter (windows); cmd-enter (mac)

EXAMPLE

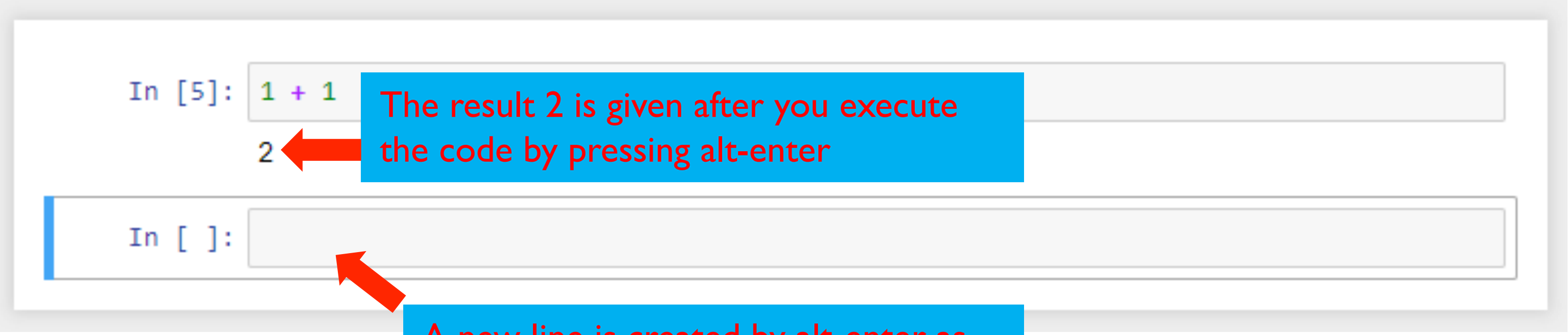
- Step 1: Type 1 + 1 into the first line



```
In [ ]: 1 + 1
```

Type 1 + 1 into this line

- Step 2: Press alt-enter to execute the line and create a new one

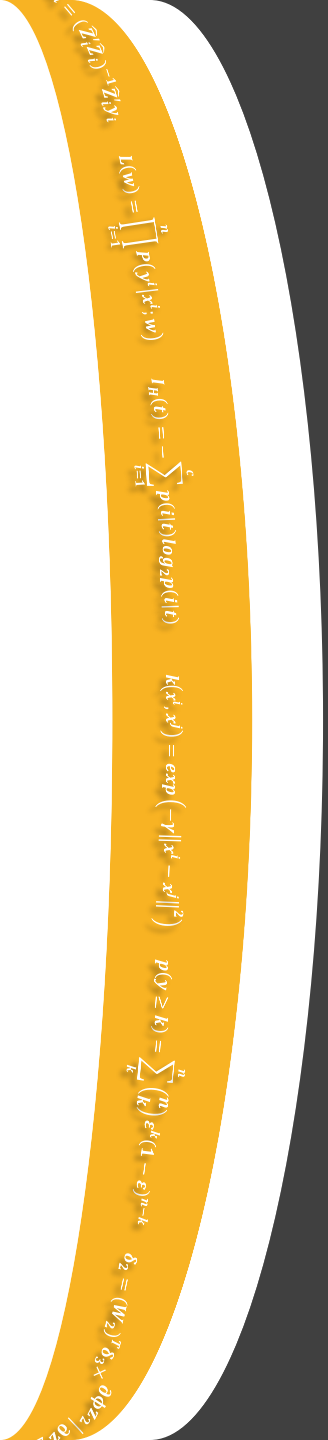


```
In [5]: 1 + 1
        2
In [ ]:
```

The result 2 is given after you execute the code by pressing alt-enter

A new line is created by alt-enter as well

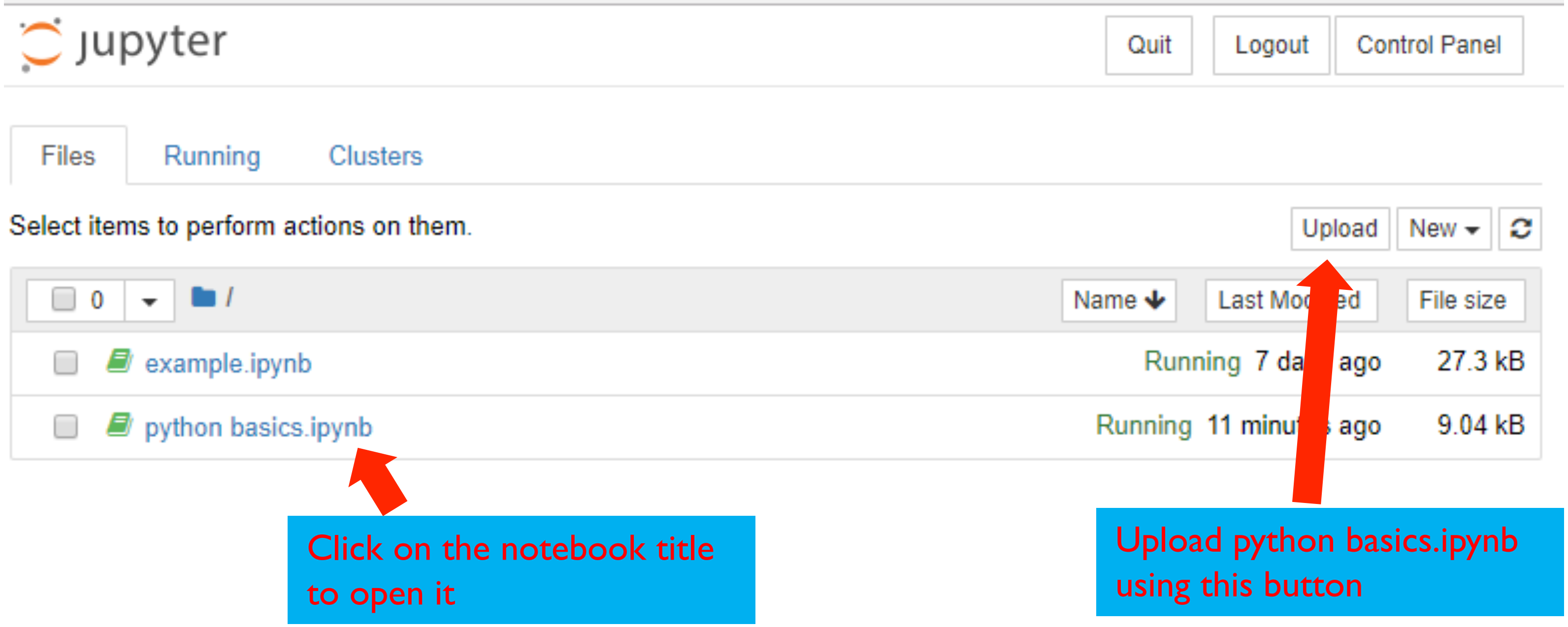
- To select a line, simply click on it


$= (Z_i'Z_i)^{-1}Z_i'y_i$
$L(w) = \prod_{i=1}^{n} P(y^i|x^i;w)$
$I_H(t) = -\sum_{i=1}^{c} p(i|t)log_2p(i|t)$
$k(x^i,x^j) = exp(-\gamma\|x^i - x^j\|^2)$
$p(y \geq k) = \sum_{k}^{n}\binom{n}{k}\varepsilon^k(1-\varepsilon)^{n-k}$
$\delta_2 = (W_2)^T\delta_3 * \partial\phi z_2 / \partial z_2$

BASIC CODING TECHNIQUES (USING PYTHON)

UPLOADING SAMPLE NOTEBOOK

- For your first hands-on coding lesson, upload python basics.ipynb into your dashboard using the upload button



Select items to perform actions on them.

Name	Last Mod ed	File size
□ example.ipynb	Running 7 da ago	27.3 kB
□ python basics.ipynb	Running 11 minu s ago	9.04 kB

Click on the notebook title to open it

Upload python basics.ipynb using this button

- Click on the notebook title to open it

BASIC CODING TECHNIQUES WITH PYTHON

- An important best practice in coding is to write comments to describe what your code does
  - Comments start with #
  - Comments will not be executed with the codes
  - Comments are not just meant for you, but for your team mates as well
  - A good piece of software typically has more comments than codes

- Follow the next few slides, supplemented by the comments and codes in “python basics.ipynb”
  - A good way to learn about these principles is to create a new line after each demonstration and try out new values for yourself and see how it works

OPERATORS

- Addition: +
- Subtraction: -
- Division: /
- Multiplication: *
- Power: **
- Modulo: %
  - modulo returns the remainder
    - 2%2 = 0 because 2 divided by 2 has a remainder of zero
    - 3%2 = 1 because 3 divided by 2 has a remainder of 1

- Greater than: >
  - Greater than or equals to: >=
- Less than: <
  - Less than or equals to: <=
- Equals: ==
- Not Equal: !=
- Not: !

DATA TYPES

- The 3 most common data types are:
  - Integers (1, 2, 3, 4, 5, …)
  - Float (1.2, 2.4, 5.66, 9.53432, …)
  - Strings ("This is a string.")

- Note that strings are enclosed in quotes " "
  - Any combination of alphabets that are not enclosed in quotes are treated as system inputs
    - no_of_apples is a variable
    - "no_of_apples" is a string (Python and R will treat it as a series of alphabet input)
  - Python and R treats "3" as a string, and 3 as an integer
  - Python and R treats "3.454" as a string, and 3.454 is a float

- Any input after # is treated as comment (Python and R will not execute these when run)
  - Use comments extensively to document and explain your codes

VARIABLES

- Variables are declared by assigning a value (integer, float or string), collection (list/array) or object (data.frame, regression results) to a sequence of alphanumeric characters
  - Note that Python and R are case-sensitive
  - price_of_apple = 20
  - Price_of_Apple = 40 (price_of_apple and Price_of_Apple are 2 different variables)
  - my_name = “Jack”
- Behaviors
  - Variables can be replaced by reassigning it with another value
    - price_of_apple = 35 (This overwrites the previous value of 20)
  - Variables can be assigned to another variable, the new variable is an independent copy of the assignor
    - original_var = 20
    - new_var = original_var
    - When you execute new_var, you get the value of 20
    - When the value of original_var is changed, new_var remains unchanged at 20

STRING FORMATTING

- The print() statement is used to show the results of operations, logic, or variables in the console
  - When you press ctrl-enter or alt-enter, the results are shown immediately in the space below the line
- Often, we want the results to be shown with some comments or information
  - john_apples = 40
  - peter_apples = 20
    - print(john_apples) will return 40 in the console
  - I can print the results with some comments like "John has 40 apples, and Peter has 20 apples" using variables instead of hardcoding the numbers
    - This is known as string formatting in python
    - The general format is string.format(variable name, variable name, ...)
      - "John has {} apples, Peter has {} apples.".format(john_apples, peter_apples)
    - The variables in .format() will replace the {} in the string in sequence

COLLECTIONS

- We always encounter collections of elements as variables in our data work
  - A collection of elements can be integer, float, strings, or a mix of these data types, separated by commas and encapsulated in brackets
    - Example: net_profit = [114.3, 98.7, 156.8, 18.3, -56.7]

- There are 4 main types of collections in Python
  - List [] :A mutable collection (i.e. elements' value can be reassigned)
  - Tuple () :An immutable collection (i.e. elements' value cannot be reassigned, its like a locked list)
  - Dictionary {key: value} :An indexed collection (the value of each element can be retrieved by dictionary[key])
  - Set set() :A collection with only unique values

SUBSETTING COLLECTIONS

- Elements can be extracted via subsetting [ ]
  - Subsetting is a practice that uses positions of elements to retrieve a slice of the collection
    - Note that different programming languages uses different starting index
    - Python uses the value of 0 for the starting index (i.e. first element in the collection), while R uses the value of 1 for the starting index
  - my_list = [1, 2, 3, 4]
  - my_list[2] will give you 3
    - The number in the [ ] is known as the index or position of the element in the array
- List/array can be sliced using the [:] method
  - my_list[1:3] will give you [2, 3]
  - The number before : is the starting index, and the number after : is the ending index (Note that python's slice does not include the ending index position, but R does)

LOGIC (IF-ELSE)

- If-else statements are used to return values given different conditions

```
if condition1:
    do this
else if condition 2:
    do this
else:
    do this
```

  - A condition is a test of True/False or Exist/Don't Exist statements
    - 3 > 2 evaluates to True, 2 < 3 evaluates to False
    - 4 in (1, 2, 3, 4) evaluates to True
    - 5 in (1, 2, 3, 4) evaluates to False

FLOW CONTROL – FOR LOOP

- Other than the if-else if-else statement, there are 2 important flow control methods:
  - For loop
  - While loop
- Loops allow us to iterate through a collection, and perform actions on each element as we iterate
- Given a list of elements, a for-loop

```
for n in my_list:
    do this to n
```

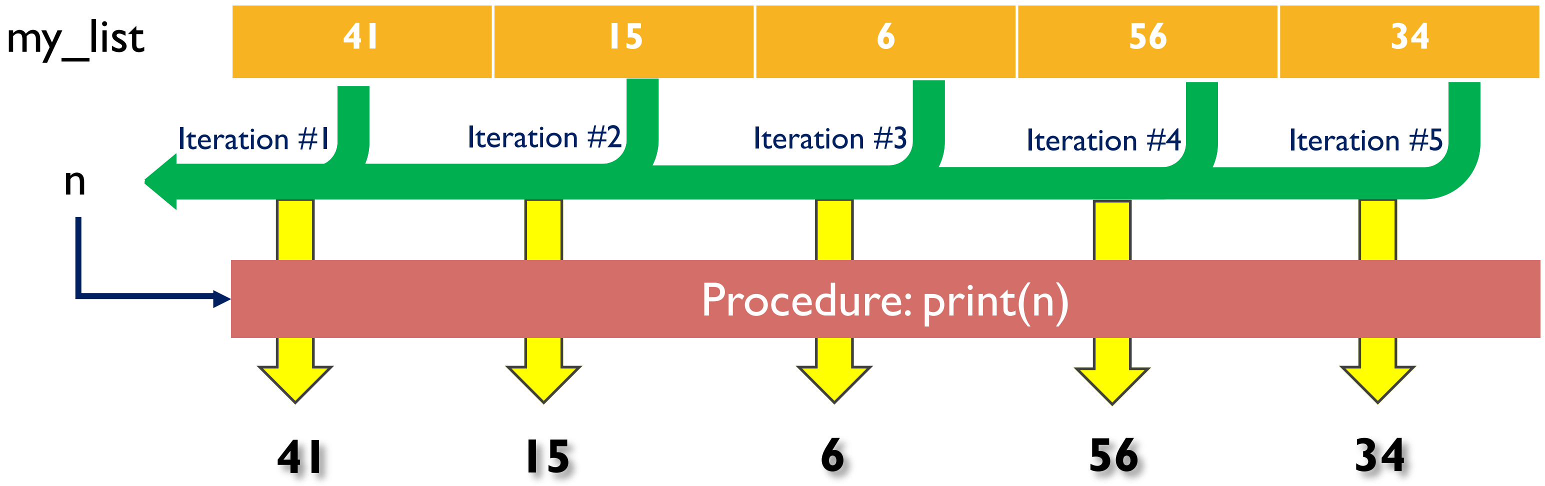
1. Assigns the value of the first element to n (you can name this anyway you want)
2. Runs the block of code under the for loop
3. After the block is done, the for loop will then assign the value of the next element to n
4. Runs the block of code again (now with the value of n being that of the 2nd element)
5. Repeat until the list is exhausted (i.e. every element in the list has been run through once)

FOR LOOP

What goes on under the hood:

```
for n in my_list:
    print(n)
```

Variables



n is used as a variable in the procedure and takes the value of each element in my_list as the for loop iterates through the list

In the first iteration, n takes the value of the first element in my_list (41). The block of code within the for loop acts on n: print(n) -> returns 41

The block of code ends, for loop goes to the next element (15) and assigns it to n

WHILE LOOP

- Similar to a For-Loop, a While-Loop is an iteration
- In a for-loop, we specify the condition as follows

```
for i in (1, 2, 'Happy', 4, 9.567):
    # do this
```

  - The loop will iterate through the list c(1, 2, 'Happy', 4, 9.567) element by element
  - For each iteration, it will assign the value of the element to i
  - After which, it will process the block of instructions in { }
- Supposed that you have 5 million elements that we wish to iterate
- In this case, we may wish to use a while loop, which has the following characteristics:
  - The while loop will keep iterating and processing the block of instructions until the condition (in this case, i < 50) is evaluated to be False
  - 2 important things:
    - Set a counter (in the e.g., i = 0 sets a counter i to 0)
    - After every block of instruction, increment the counter by 1

```
i = 0
while i < 50:
    # do this
    i = i + 1
```

FUNCTIONS

- Many a times we would want to re-use some of the instructions that we have coded before
- And as we re-use the instructions for each subsequent cases, there are variable values that we would like to change
- Instead of cutting and pasting the same set of codes throughout your script, and manually and painstakingly change the variable values for each case, we can create a function to improve our productivity
  - The other benefit of writing a function, is that we only need to amend or update the instructions (typically computation algorithms/models) once
  - The alternative is to find every instance of the instructions and manually cut and paste to update them
    - Very error prone

PRACTICE QUESTIONS

- Variable assignment
  - Peter went to a fair and won 10 marbles
  - He played a game with Gina and lost 4 marbles
  - As a result, Gina has twice more marbles than Peter
  - She then gave 2 marbles to Tim
  - How many marbles does Gina have after that?

- If/else
  - If the number of marbles that Gina has is even, print "Gina has an even number of marbles."
  - If not, print "Gina has an odd number of marbles."

PRACTICE QUESTIONS

- Given the following lists
  - years = [2010, 2011, 2012, 2013, 2014, 2015]
  - net_profit = [27.5, -54.7, 4.6, 13.2, -25.6, 45.8]
- Do the following:
  - Convert the above list into a dictionary
  - Create the following 2 functions and apply them as a list comprehension:
    - Returns the year in which profits were made, else 0
      - E.g. [2010, 0, 2012, 2013, 0, 2015]
    - Returns the amount of profits they needed to breakeven in a loss year
      - E.g. [0, 54.7, 0, 0, 25.6, 0]

INTRODUCTION TO R PROGRAMMING

BASICS OF R

- The basics of R coding is very similar to the techniques we have learnt in Python in the previous segment
- To start with this segment, upload “r basics.ipynb” into your dashboard
- Go through the guide from the next slide onwards, supplemented with the codes in “r basics.ipynb”

OPERATORS

(NOTE THE DIFFERENCE BETWEEN R AND PYTHON IN POWER AND MODULO)

- Addition: +
- Subtraction: -
- Division: /
- Multiplication: *
- Power: ** or ^
- Modulo: %%
  - modulo returns the remainder
    - 2%%2 = 0 because 2 divided by 2 has a remainder of zero
    - 3%%2 = 1 because 3 divided by 2 has a remainder of 1

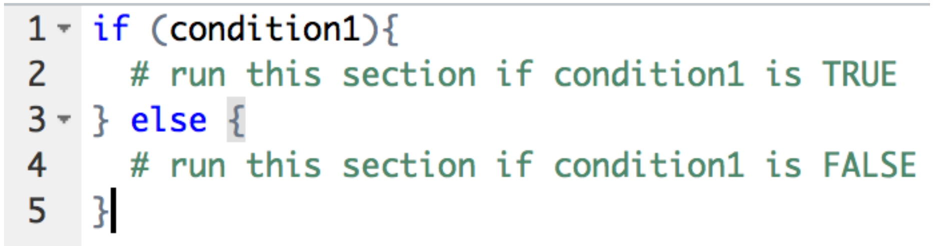
- Greater than: >
  - Greater than or equals to: >=
- Less than: <
  - Less than or equals to: <=
- Equals: ==
- Not Equal: !=
- Not: !

VARIABLES

- Variables in R are declared the same way as in Python
- Variables in R also behave similarly
  - Variables can be replaced by reassigning it with another value
    - price_of_apple = 35 (This overwrites the previous value of 20)
  - Variables can be removed with the command: remove(price_of_apple)
    - To completely clear the memory used by this variable, execute the command: gc()
    - gc stands for ”garbage collector”
  - Variables can be assigned to another variable, the new variable is an independent copy of the assignor
    - original_var = 20
    - new_var = original_var
    - When you run new_var, you get the value of 20
    - When the value of original_var is changed, new_var remains unchanged at 20

LOGIC (IF-ELSE)

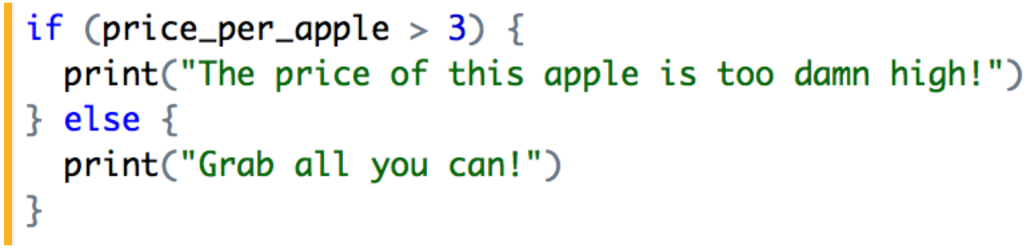
- Simple if-else statement



```
if (condition1){
  # run this section if condition1 is TRUE
} else {
  # run this section if condition1 is FALSE
}
```

  - A condition is a test of TRUE/FALSE statements
    - 4 > 3 evaluates to TRUE
    - ‘happy’ %in% c(‘I’, ‘am’, ‘so’, ‘sad’) evaluates to FALSE
    - try them out in the notebook

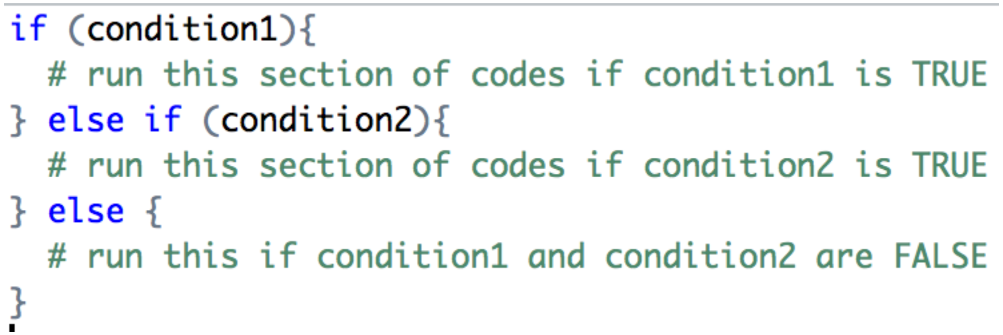
Try this out in the console as well:



```
if (price_per_apple > 3) {
  print("The price of this apple is too damn high!")
} else {
  print("Grab all you can!")
}
```

LOGIC (IF-ELSEIF-ELSE)

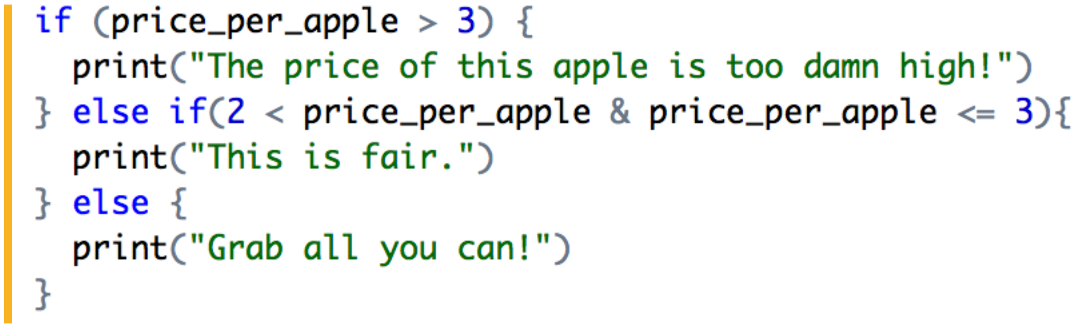
- if-elseif-else statement



```
if (condition1){
  # run this section of codes if condition1 is TRUE
} else if (condition2){
  # run this section of codes if condition2 is TRUE
} else {
  # run this if condition1 and condition2 are FALSE
}
```

  - Used when our logic has more than 1 condition
  - The else segment will capture "everything else" that the preceding conditions fail to capture

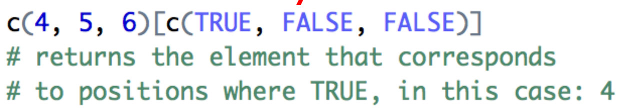
Try this out in the console as well:



```
if (price_per_apple > 3) {
  print("The price of this apple is too damn high!")
} else if(2 < price_per_apple & price_per_apple <= 3){
  print("This is fair.")
} else {
  print("Grab all you can!")
}
```

DATA STRUCTURES I

- A collection of elements (can be integer, float or strings) is known as a list or array
  - In R, we specify this as c(…) and the elements are separated by commas
    - E.g. c(1, 2, 3, 4)
  - A list/array can contain mixed datatypes
    - c(23, "hello kitty", 4.564) is a valid list
  - Single element or collections can be added together by nesting them in another c()
    - c(c(1, 2, 3, 4), 56) will give you c(1, 2, 3, 4, 56)
  - Elements can be extracted via subsetting [ ]
    - c(1, 2, 3, 4)[2] will give you 2
    - The number in the [ ] is known as the index or position of the element in the array
      - The first element in the array has an index of 1 (Note that python starts with 0)
  - List/array can be sliced using the [:] method
    - c(1, 2, 3, 4)[2:3] will give you c(2, 3)
    - The number before : is the starting index, and the number after : is the ending index (Note that python's slice does not include the ending index position)
  - List/array can be sliced via subsetting a TRUE/FALSE array as well



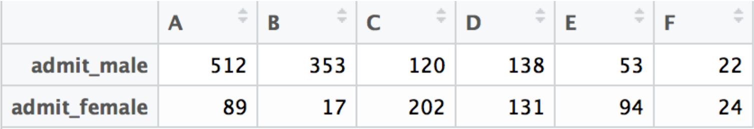
```
c(4, 5, 6)[c(TRUE, FALSE, FALSE)]
# returns the element that corresponds
# to positions where TRUE, in this case: 4
```

INBUILT FUNCTIONS

- Functions are commands and instructions packed into a token with parenthesis (), such as na.omit() and scatterplot()
  - Functions help in 2 ways:
    - Allows easy reusability of procedures
    - Allows easy replacement of values
- R and its 3rd-party libraries come with tons of pre-built functions
  - To use a function, input your parameters into the parenthesis
  - E.g. for the max function, suppose we have a list of values c(3, 4, 5, 6, 7, 8)
    - We first assign the values to a variable called myValues
      - myValues = c(3, 4, 5, 6, 7, 8)
    - max(myValues) will return 8
  - Look through the R documentation (e.g. use ?max) to find out what parameters are required for the function to work as intended

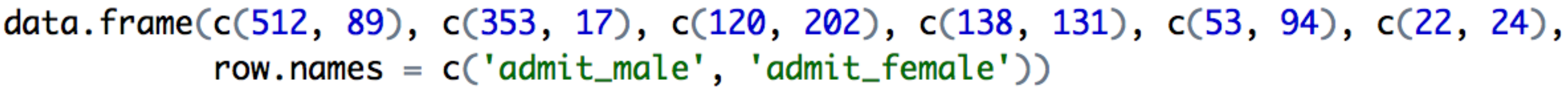
DATA STRUCTURES II

- A dataframe looks like a table with rows and columns



	A	B	C	D	E	F
admit_male	512	353	120	138	53	22
admit_female	89	17	202	131	94	24

- While list/arrays are 1-dimensional, dataframes are 2 dimensional (rows and columns)
  - However, manipulating a dataframe is similar to that of a list/array
  - Just need to remember that we are dealing with columns in addition to rows
- Dataframes can be constructed from list/arrays
  - Each list of values will form one column
  - In the above example, the dataframe is constructed by:

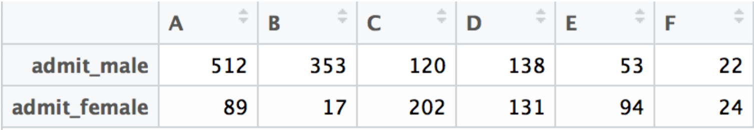


```
data.frame(c(512, 89), c(353, 17), c(120, 202), c(138, 131), c(53, 94), c(22, 24),
           row.names = c('admit_male', 'admit_female'))
```

- Dataframes can be subsetted with [row, col]
  - dataframe[1:1, 4:5] will return the first row with columns D and E
    - From the comma, omitting everything to the left will return all rows (i.e. [, 4:5]), omitting everything to the right will return all columns (i.e. [1:1,]
  - Idiosyncracies that will happen if comma is omitted:
    - dataframe[1] will return first column
    - dataframe[1:2] will return first 2 columns

DATAFRAMES

- Dataframes is the object that we will work with the most



	A	B	C	D	E	F
admit_male	512	353	120	138	53	22
admit_female	89	17	202	131	94	24

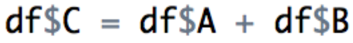
(Suppose we have assigned the above dataframe to a variable named admit)

- There are some idiosyncracies that we have to take note
  - We can extract a column of values from the dataframe using a single square bracket with the column name
    - admit['A'] will return the first column of values 512, 89
    - However, this single square bracket operation returns a dataframe
  - Probem 1: Some R functions (e.g. levels) cannot work with dataframe even if its only 1 column
    - Many other functions do not have this issue (e.g. max, sum)
  - Problem 2: Dataframe cannot be subsetted using a TRUE/FALSE array (but can be subsetted with a 1-column dataframe of TRUE/FALSE)
  - In these cases, we need to use a double square bracket [[ ]] to extract the column of values as a list/array (Note: admit$A returns list/array, not dataframe)
    - The difference between single and double square brackets is that the former returns a dataframe (which some functions cannot use) while the latter returns a list/array

DATAFRAME/ARRAY OPERATIONS

- Array and matrix operations and procedures are optimized
  - R and Python scientific stack processes dataframes array by array
  - Test of conditions between 2 array returns an array of TRUE/FALSE
  - Subsetting arrays with conditions returns an array of elements whose results correspond to TRUE

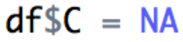
Array operations



```
df$C = df$A + df$B
```

df$C		df$A		df$B
C		A		B
35		2		33
47		2		45
15	=	4	+	11
95		8		87
102		7		95
38		5		33

Array operations

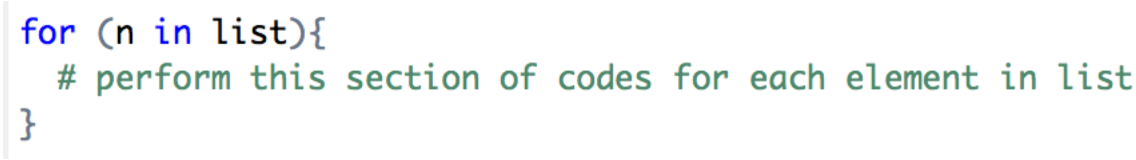


```
df$C = NA
```

df$C		df$C
C		C
35		NA
47	Reassigned with NA	NA
15	→	NA
95		NA
102		NA
38		NA

FLOW CONTROL – FOR LOOP

- 2 critical flow control methods:
  - For loop
  - While loop
- For Loop

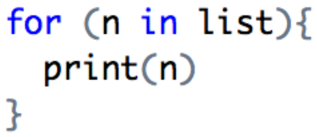


```
for (n in list){
  # perform this section of codes for each element in list
}
```

  - Given a list of elements, a for-loop
    1. Assigns the value of the first element to n (you can name this anyway you want)
    2. Runs the section of code (if the variable n is used in this set of codes, it will use the value of the first element)
    3. After it is done with the section of code { }, it will then assign the value of the next element to n
    4. Runs the section of code again (now with the value of n being that of the 2nd element)
    5. Repeat until the list is exhausted (i.e. every element in the list has been run through once)

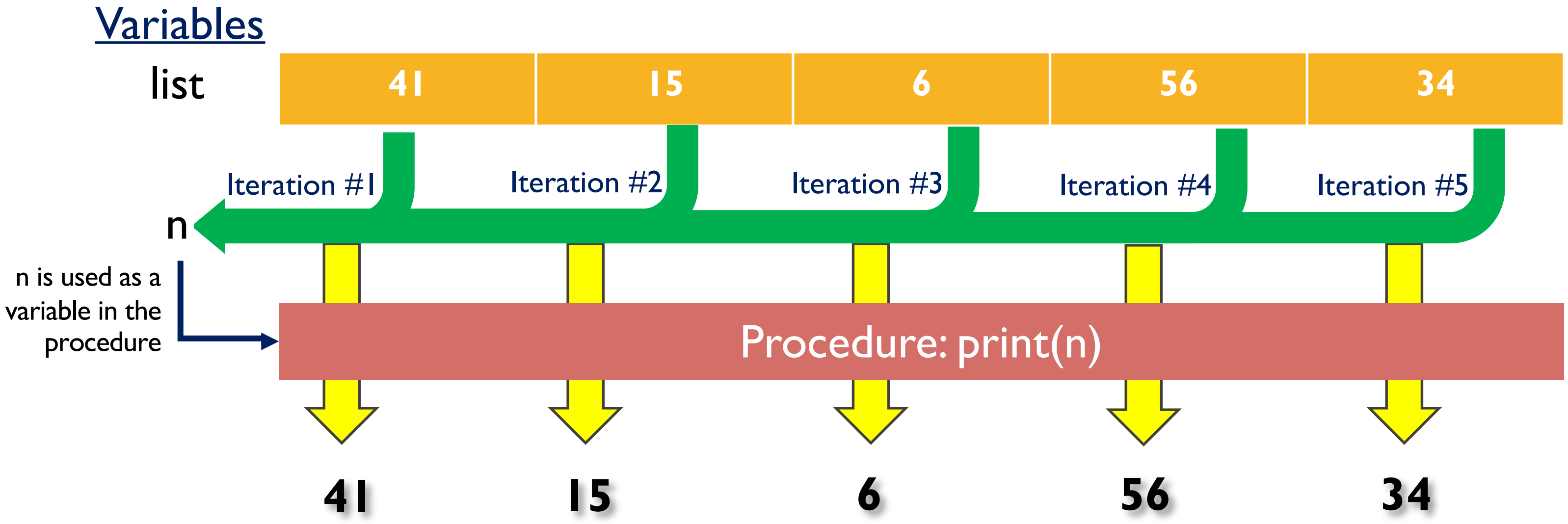
FOR LOOP

The code:



```
for (n in list){
  print(n)
}
```

What goes on under the hood:



Variables

list

41	15	6	56	34

n

n is used as a variable in the procedure

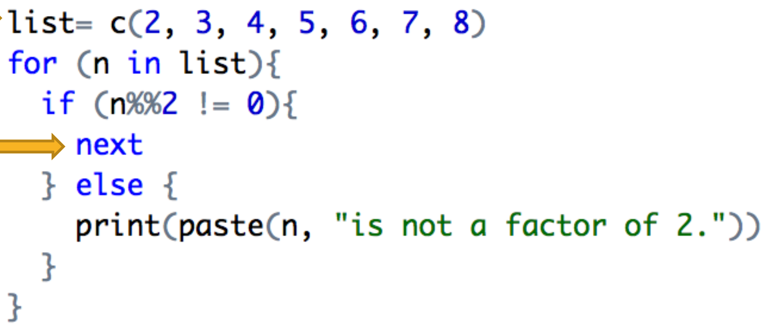
Procedure: print(n)

FOR LOOP

- We often integrate if-else with for-loop as well
  - If we want to use an if-else statement on multiple variables, and do not want to code the same set of if-else statement for each variable (which will result in a lengthy set of codes that keeps repeating itself)
  - Sometimes we want to keep the results of the executed procedures on some elements but not others
    - Use the ”next” command to proceed to the next element in the list without going through the rest of the code

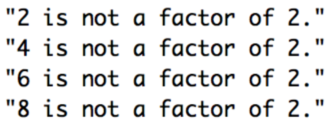
Assign the array of values to the variable list

If the remainder of n / 2 is not 0, ignore the rest of the code and start again with the next element in list



```
list= c(2, 3, 4, 5, 6, 7, 8)
for (n in list){
  if (n%%2 != 0){
    next
  } else {
    print(paste(n, "is not a factor of 2."))
  }
}
```

Results in console:

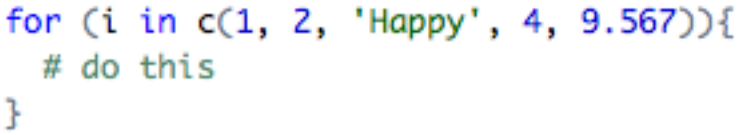


```
"2 is not a factor of 2."
"4 is not a factor of 2."
"6 is not a factor of 2."
"8 is not a factor of 2."
```

- For-loops can be nested as well, often used in cases when we want to get all the interacted results from 2 or more lists

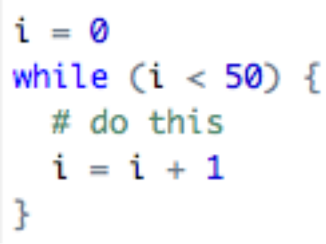
WHILE LOOP

- Similar to a For-Loop, a While-Loop is an iteration
- In a for-loop, we specify the condition as follows



```
for (i in c(1, 2, 'Happy', 4, 9.567)){
  # do this
}
```

  - The loop will iterate through the list c(1, 2,'Happy', 4, 9.567) element by element
  - For each iteration, it will assign the value of the element to i
  - After which, it will process the block of instructions in { }
- Supposed that you have 5 million elements that we wish to iterate
  - Our computer's ram will be reduced by a large amount if we load the elements into a list and applying a for-loop
- In this case, we may wish to use a while loop, which has the following characteristics:
  - The while loop will keep iterating and processing the block of instructions until the condition (in this case, i < 50) is evaluated to be FALSE
  - 2 important things:
    - Set a counter (in the e.g., i = 0 sets a counter i to 0)
    - After every block of instruction, increment the counter by 1



```
i = 0
while (i < 50) {
  # do this
  i = i + 1
}
```

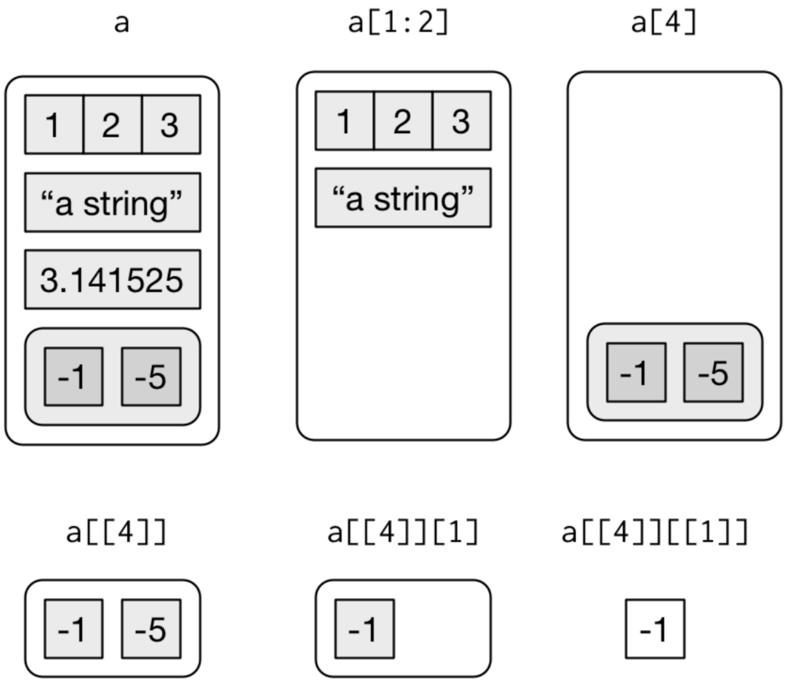
BRACKETS IN R

- ()
  - Round brackets are used to encapsulate arguments (including conditions)
  - E.g. for function max, it takes a list as an argument and returns the largest value in the list
    - max(args), where args = arguments
  - E.g. for if-else statements, where we specify a condition
    - if (a == 6){do this}

- {}
  - Curly brackets are used to encapsulate a block of instruction
  - E.g. for if-else statements: if (a ==6) {do this}
  - Curly brackets allow you to write multiple lines within the block

BRACKETS IN R

- [ ] Square brackets are used for subsetting data objects (such as list and dataframes)
- R abstracts the [] operation by allowing two methods
  - [pos] returns the element in the data object by its position in the object (e.g. c(1, 0, 4, 5)[3] returns the value of the 3rd element: 4)
  - [row, column] returns a slice of the data object specified by start:row (e.g. [1:3,] returns rows 1 to 3 with all columns) Note: empty value implies return all
  - Note that [] returns elements, so if a list is nested (e.g. list(c(1, 2, 3, 4), c(5, 6, 7, 8))), then [1] will return the value of the first element which is c(1, 2, 3, 4)
- [[ ]] Double square brackets are used to extract a single component from a list AND REMOVES THE HIERARCHY
  - This means that if used on a dataframe (e.g. df[[gender]]), it will return a list of values for the column gender.
- The shorthand for this is $ (e.g. df$gender is the same as df[[gender]])

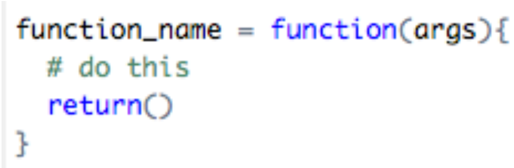


FUNCTIONS I

- Many a times we would want to re-use some of the instructions that we have coded before
- And as we re-use the instructions for each subsequent cases, there are variable values that we would like to change
- Instead of cutting and pasting the same set of codes throughout your script, and manually and painstakingly change the variable values for each case, we can create a function to improve our productivity
  - The other benefit of writing a function, is that we only need to amend or update the instructions (typically computation algorithms/models) once
  - The alternative is to find every instance of the instructions and manually cut and paste to update them
    - Very error prone

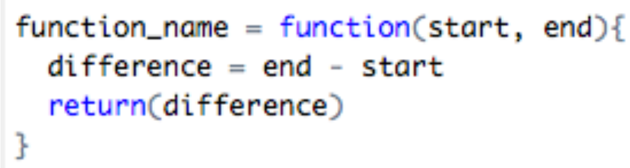
FUNCTIONS II

- We have been using inbuilt functions in R
- max() is such a function, which returns us the element with the largest value
  - max(c(4, 3, 7, 6, 4, 5, 7, 9)) returns 9
- The basic format of a function is as follows:



```
function_name = function(args){
  # do this
  return()
}
```

  - Declare a function name (function_name) as a variable
  - Assign it with a function object (function())
  - Depending on the variables that you want to use in the block of instructions, you can name them as arguments (you can have tons of them if you need)

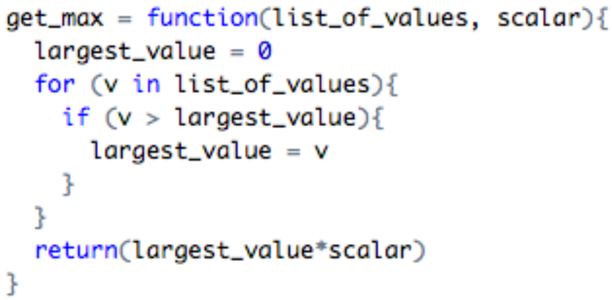


```
function_name = function(start, end){
  difference = end - start
  return(difference)
}
```

  - A return statement to specify the result of the function (in the above instance, running the function will return the difference between start and end
- Calling the function is simply putting in the arguments and executing it (in this case, function_name(4, 8) will return a value of 4)

FUNCTIONS III

- Let’s try building a custom max function
- We want this function to return the largest value of a list, and multiply it by a scalar
  - The list and scalar can differ in all our cases



```
get_max = function(list_of_values, scalar){
  largest_value = 0
  for (v in list_of_values){
    if (v > largest_value){
      largest_value = v
    }
  }
  return(largest_value*scalar)
}
```

- The setup:
  - Name the function as get_max
  - Specify the arguments (in this case 2: list_of_values and scalar)
    - These will be referenced in the block of instructions to return different results from different arguments
- The logic
  - Create a variable (largest_value) to store the largest value, set the initial value to 0
  - Use a for loop to iterate through the argument (list_of_values)
  - Replace largest_value with the element’s value if it’s larger than the existing value of largest_value
  - Once the iteration is complete, we would have the largest value in the list
  - Before returning the result, we multiply largest_value by the scalar

ADDITIONAL COMMENTS

ADDITIONAL COMMENTS

- Familiarize yourself with the basic characteristics of what was covered
- Need to know what goes on under the hood
  - Datatypes and collections
  - Techniques and idiosyncracies
- Subsequently, its all about how you exercise your creativity to get the results you want
- Practise, practise and practise!
  - Upload "r exercises.ipynb" and try out the exercises

QUESTIONS?

Email any queries to

jackhong@smu.edu.sg